Bioinformatics, YYYY, 0–0
doi: 10.1093/bioinformatics/xxxxx
Advance Access Publication Date: DD Month YYYY
Application Note

Bioinformatics, Volume 36, Issue 3, February 2020, pp. 934–935
DOI:10.1093/bioinformatics/btz634

Genome Analysis

Tersect: a set theoretical utility for exploring sequence variant data.

Tomasz J Kurowski[1], Fady Mohareb[1*]

[1]The Bioinformatics Group, School of Water, Energy and Environment, Cranfield University, College Road, Bedford, MK43 0AL, UK.

*To whom correspondence should be addressed.

Abstract

Summary: Comparing genomic features among a large panel of individuals across the same species is considered nowadays a core part of the bioinformatics analyses. This typically involves a series of complex theoretical expressions to compare, intersect, extract symmetric differences between individuals within a large set of genotypes. Several publically available tools are capable of performing such tasks, however, due to the sheer size of variants being queried, such tasks can be computationally expensive with a runtime ranging from few minutes up to several hours depending on the dataset size. This makes existing tools unsuitable for interactive data query or as part of genomic data visualization platforms such as genome browsers. Tersect is a lightweight, high-performance command-line utility which interprets and applies flexible set theoretical expressions to sets of sequence variant data. It can be used both for interactive data exploration and as part of a larger pipeline thanks to its highly optimized storage and indexing algorithms for variant data.

Availability and implementation: Tersect was implemented in C and released under the MIT licence. Tersect is freely available at https://github.com/tomkurowski/tersect.

Contact: f.mohareb@cranfield.ac.uk.

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Large scale genome resequencing projects such as the 100,000 genome project (Turnbull et al. 2018), the 1000 Genomes Project (Genomes Project et al. 2015) or the 150 Tomato Genome Re-Sequencing Project (Tomato Genome Sequencing et al. 2014) provide researchers with large-scale references for genetic variation. These can be compared with novel data to help identify causal variants and QTLs, delimit haplotype blocks and introgressions, or infer phylogenetic relationships (Gao et al. 2019). All of those uses require means of filtering and comparing the variant contents between large phenotypic groups in order to identify concordant and discordant variants. These can be considered applications of set theoretical operations such as intersections or unions on sets of variants.

While multiple tools such as BEDOPS (Neph et al. 2012), BCFtools (Danecek et al. 2011), and BEDTools (Quinlan and Hall 2010) offer the option to execute such operations, they are relatively inflexible in the complexity of possible queries and rely on parsing input files as they are executed, limiting their speed and responsiveness.

We hereby present Tersect, a tool which allows users to construct queries of any level of complexity by providing its own declarative query language and significantly speeds up their execution using specialized bitmap indices. Queries are interpreted, optimized, and executed in a single step, either on entire genomes or on selected genomic regions, making the process extremely fast and responsive, ideal for an exploratory approach to investigating genome contents.

2 Tersect

Tersect imports VCF file data and uses bitmap indexing to encode binary information on the presence or absence of specific variants in each individual genome while building up a single unified database of alleles across all collected genomes.

The database is sorted and indexed by position and identity. When traversed in order, the stored list of variants is parallel to the per-genome bitmap indices, linking the two data structures. The index on variants and their positions allows for rapid identification of regions of interest by chromosome and position range, while the bitmap indices allow for highly efficient comparisons between genomes, leveraging bitwise operations to compare many sites at once. As any given genome contains only a relatively small subset of possible alleles, the bitmaps are sparse and easily compressed. Tersect uses a variant of the Word-Aligned Hybrid (WAH)

lossless compression method (Wu et al. 2006) which allows logical operations without an explicit decompression step (See Supplementary Data S2 for detailed outline of Tersect storage architecture).

The variant data and indices are stored in a special index file which only needs to be generated once per collection of genomes and can be shared and used independently of the source data. Tersect uses a memory-mapped I/O approach to access index file contents, allowing for random access to regions of interest and limiting the memory footprint of queries.

A disadvantage of bitmap indexing, shared by Tersect, is the relative inefficiency of updating and adding data. This indexing approach is generally best suited to read-only applications and is often used in data warehousing. However, the stored data (allele identities and presence in genomes) do not frequently change over time and are generally added in batches (at least one genome at a time), mitigating such disadvantages. The Tersect index builder uses a highly efficient, priority-queue-based merge method, allowing for an index file to be rapidly re-created.

Tersect features a command parser which allows a user to enter set theoretical expressions operating on genomes (as sets) and variants (as set elements) and including set theoretical operations such as intersections, unions, and symmetric differences (See Supplementary Data S1). These can be arranged into queries of arbitrary complexity, including deeply nested expressions, using a simple syntax. Rather than merely executing the parsed operations in sequence, Tersect builds an abstract syntax tree (AST) which represents the entered expression. The tree can then be optimized to simplify and speed up operations. If the user requests data from multiple genomic regions using the same command, the same AST is reused for each.

3 Results and discussion

Tersect was benchmarked against three tools which offer similar functionalities: BCFtools v.1.9, BEDTools, and BEDOPS. It should be noted that, as they are designed to compare variant sets not only to each other but also to other types of data, the last two tools focus on *positional* overlap and intersection between features rather than variant identity. This means that overlapping but distinct variants, such as different alleles at multi-allelic sites or InDels which span across SNV sites, are considered to be intersecting. This leads to subtly different results delivered by the tested tools.

The data used for comparison were publicly available tomato genomes from two studies (Dinh et al. 2018; Lin et al. 2014; Tomato Genome Sequencing et al. 2014), for a total of 444 re-sequenced genomes of tomato cultivars and closely related species.

Two important shared functionalities were tested: the identification of private variants, that is variants occurring only in a single specific genome out of a collection of genomes, and the intersection of a group of genomes to identify variants shared by each of them (also known as *concordant* variants). For the former test, subsets of the 444 genomes collection were used. For the latter, subsets of 56 *S. pimpinellifolium* genomes which contain large regions of shared variation distinct from the *S. lycopersicum* reference were used. Input data for each of the tools were converted into the most appropriate format (e.g. BED for BEDTools) and indexed (where appropriate) prior to the benchmarking. This also applies to Tersect, as the time taken to build an index, which needs to be done only once, was not included in the test runtimes. Index file generation for the largest genome collection (444 genomes) took ten minutes (Supplementary Data S3 - Figure 1 and Table 3), which is fast enough to make Tersect the fastest tool even if indexing time were to be included in the benchmark.

It can be seen that Tersect performs from three to over a hundred times faster than BCFtools, which is the fastest of the other three applications (see Figure 1 and Supplementary Figure 2). The difference is more pronounced for larger inputs and this trend is likely to continue for data sets larger than those examined in this paper. This presents a promising outlook for the scalability and future usability of Tersect as more genomes are re-sequenced every year and the volume of available data continues to rapidly increase. The runtime of all four tools follows a roughly linear relationship with the size of the input. The superior speed of Tersect stems from the highly problem-specific optimization and indexing scheme rather than from improved algorithmic time complexity in the strict sense.

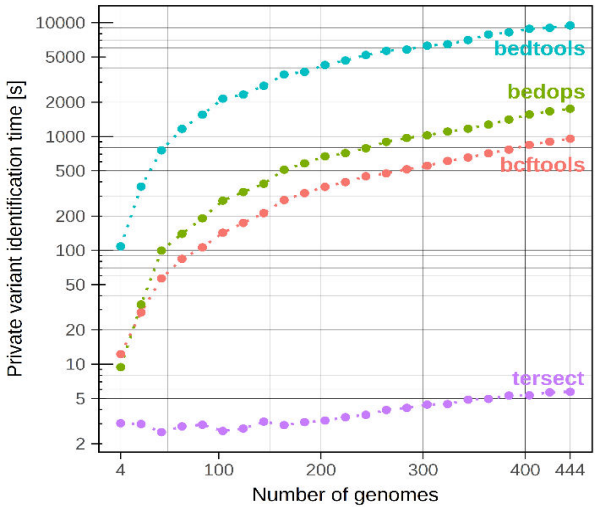


Fig. 1. Benchmarking results for the identification of variants private to a single genome out of subsets of 444 tomato genomes. All four applications show a linear relationship between the input size (number of genomes) and execution time; for Tersect this relationship is partially obscured by the relatively slow disk read / write operations which comprise a significant proportion of the runtime, especially for small input sizes. See Supplementary Table S1 for the numeric results.

Tersect is the only among the evaluated tools capable of executing complex queries on large real-world data sets in a matter of seconds, making this the tool of choice to be used interactively, rather than as part of a batch processing pipeline. In combination with the flexible query syntax, this high performance offers new possibilities for real-time, exploratory use of the ever-growing volume of genomic data being produced today.

Funding

Tersect was developed as part of the BBSRC-funded project BB/L011611/1.

References

Danecek, P., et al. (2011), 'The variant call format and VCFtools', *Bioinformatics,* 27 (15), 2156-8.

Dinh, Q. D., et al. (2018), 'Exploring natural genetic variation in tomato sucrose synthases on the basis of increased kinetic properties', *PLoS One,* 13 (10), e0206636.

Gao, L., et al. (2019), 'The tomato pan-genome uncovers new genes and a rare allele regulating fruit flavor', *Nat Genet,* 51 (6), 1044-51.

Genomes Project, Consortium, et al. (2015), 'A global reference for human genetic variation', *Nature,* 526 (7571), 68-74.

Lin, T., et al. (2014), 'Genomic analyses provide insights into the history of tomato breeding', *Nat Genet,* 46 (11), 1220-6.

Neph, S., et al. (2012), 'BEDOPS: high-performance genomic feature operations', *Bioinformatics,* 28 (14), 1919-20.

Quinlan, A. R. and Hall, I. M. (2010), 'BEDTools: a flexible suite of utilities for comparing genomic features', *Bioinformatics,* 26 (6), 841-2.

Tomato Genome Sequencing, Consortium, et al. (2014), 'Exploring genetic variation in the tomato (Solanum section Lycopersicon) clade by whole-genome sequencing', *Plant J,* 80 (1), 136-48.

Turnbull, C., et al. (2018), 'The 100 000 Genomes Project: bringing whole genome sequencing to the NHS', *BMJ,* 361, k1687.

Wu, Kesheng, Otoo, Ekow J., and Shoshani, Arie (2006), 'Optimizing bitmap indices with efficient compression', *ACM Trans. Database Syst.,* 31 (1), 1-38.

Tersect: a set theoretical utility for exploring sequence variant data.

Tomasz J Kurowski[1], Fady Mohareb[1*]

[1] The Bioinformatics Group, School of Water, Energy and Environment, Cranfield University, College Road, Bedford, MK43 0AL, UK.

[*] To whom correspondence should be addressed.

Contact: f.mohareb@cranfield.ac.uk.

Supplementary Material S1: User Manual

Table of Contents

Background

Tersect is a command-line utility for conducting fast set theoretical operations and genetic distance estimation on biological seque$

Tersect is intended to allow for highly responsive, exploratory interaction with variant data as well as for integration with larger$

Tersect can also be used to provide estimates of genetic distance between sets of samples, using the number of differing sites as a $

Installation

Pre-compiled releases

Tersect packages and binaries are available for download below:

Linux

- 64-bit binaries: tersect-0.12.0-Linux.tar.gz
- 64-bit .deb package (Debian, Ubuntu): tersect-0.12.0-Linux.deb
- 64-bit .rpm package (Fedora, openSUSE): tersect-0.12.0-Linux.rpm

macOS

- 64-bit binaries: tersect-0.12.0-macOS.tar.gz

Building Tersect from source

Building Tersect from source requires CMake version 3.1+ as well as Flex (lexical analyzer) version 2.5+ and Bison (parser generator) version 2.6+.

1. Cloning the repository

```
git clone https://github.com/tomkurowski/tersect.git
```

2. Building

For an out-of-source build after cloning the repository use the following commands:

```
cd tersect
mkdir build
cd build
```

```
cmake ..
make
```

3. Installing

This step may require elevated permissions (e.g. prefacing the command with `sudo`). The default installation location for Tersect is `/usr/local/bin`.

```
make install
```

Example data

Two archives containing example Tersect index files (.tsi) are available for download below to allow you to try out the utility without needing to create an index file yourself.

The first index contains human genomic variant data for 2504 individuals from the 1000 Genomes Project. While Tersect is capable of handling the entire human genome, the index below is limited to chromosome 1 to make the example archive smaller and quicker to download.

The second index contains tomato genomic variant data for 360 tomato accessions from the AGIS Tomato 360 Resequencing Project and 84 accessions from the Wageningen UR 150 Tomato Genome Resequencing Project for a combined data set of 444 accessions. Samples have been renamed according to a provided key (accession_names.tsv) to make them more informative and consistent between the two source data sets.

Note: the index files provided below are compressed using gzip and need to be uncompressed before use.

- 2504 human genomes, chromosome 1
- 444 tomato genomes [sample names: accession_names.tsv]

Building a Tersect index

You can build your own Tersect index based on a set of VCF files using the `tersect build` command. You need to provide a name for your index file (a .tsi extension will be added if you omit it) as the first argument, followed by any number of input VCF files (which may be compressed using gzip) to be included in the index. Please note that although from a technical point of you, Tersect would work even if your VCF files were called against different reference genomes or versions of the same reference, the biological context of your theoretical operations won't be accurate (depending on how different the reference genomes used). Therefore, we strongly recommend using VCF files called against the same reference version.

Example:

The command below builds a Tersect index file named *tomato.tsi* which includes variants from all *.vcf.gz* files in the *data* directory. Depending on the input size this can take several minutes.

```
foo@bar:~$ tersect build \
           tomato.tsi ./data/*.vcf.gz
```

Optionally, you can also provide a `--name-file` input file containing custom sample names to be used by Tersect. These names will replace the default sample IDs defined in the input VCF header lines. The `--name-file` should be a tab-delimited file containing two columns, the first with the sample IDs to be replaced and the second with the names to be used by Tersect. An example is shown below:

```
TS-1    S.lyc B TS-1
TS-10   S.lyc B TS-10
TS-100  S.lyc B TS-100
TS-101  S.lyc B TS-101
TS-102  S.lyc B TS-102
TS-103  S.lyc B TS-103
TS-104  S.lyc B TS-104
TS-108  S.lyc B TS-108
TS-11   S.lyc B TS-11
TS-110  S.lyc B TS-110
```

You can also modify sample names in an existing Tersect index file by using the `tersect rename` command.

It is worth noting that the descriptive fields of the VCF files are not stored within the Tersect database. The reason for that is once an operation is performed on two of more VCF files, these fields will be discarded anyway as they are genotype-specific. However, you should be able to retrieve it back by intesecting Tersect's output with any VCF files from this list.

Inspecting a Tersect index

The data contained in a Tersect index file can be inspected using several commands. The `tersect chroms` command prints information on the number of variants present in each of the reference chromosomes as well as the chromosome names and size. **Note:** In the absence of a reference file, the *length* of each chromosome is represented by the position of the last variant, which will always be an underestimate.

Example:

The command below prints the per-chromosome variant content of the example Tersect index file named *tomato.tsi* (you can download it here).

```
foo@bar:~$ tersect chroms tomato.tsi
Chromosome  Length  Variants
SL2.50ch00  21805702    1343815
SL2.50ch01  98543411    9965680
SL2.50ch02  55340384    5189338
SL2.50ch03  70787603    6741448
SL2.50ch04  66470926    7257520
SL2.50ch05  65875078    6830857
SL2.50ch06  49751619    4870941
SL2.50ch07  68044764    6868152
SL2.50ch08  65866627    6504025
SL2.50ch09  72481975    7102356
SL2.50ch10  65527500    6712293
SL2.50ch11  56302478    5367032
SL2.50ch12  67145147    6719621
```

The `tersect samples` command prints the names of samples present in a Tersect index file. These can be either all samples or a subset based on a naming pattern (the `--match` parameter) and/or on the presence of specific variants (the `--contains` parameter).

Sample name patterns can include wildcard symbols (*) which match zero or more characters. For example, a pattern like "S.lyc*" will match all samples whose names begin with "S.lyc". A lone wildcard character matches all samples stored in the Tersect index file.

If you specify a list of variants via the `--contains` paramter, only samples which contain each of those variants will be printed. The variant format should look as follows: `chromosome:position:ref:alt` where `ref` and `alt` are reference and alternate alleles. Multiple variants can be included, separated by commas (e.g. `chr1:1245:A:G,chr8:5300:T:A`).

Examples:

The command below prints the names of samples matching the *"S.gal*"* wildcard pattern contained in the example Tersect index file *tomato.tsi*.

```
foo@bar:~$ tersect samples \
           tomato.tsi -m "S.gal*"
Sample
S.gal W TS-208
S.gal LA1044
S.gal LA1401
S.gal LA0483
```

The command below prints the names of all samples containing both a T/G SNV at position 100642 on chromosome 3 and an A/G SNV at position 5001015 on chromosome 6 contained in the example Tersect index file *tomato.tsi*.

```
foo@bar:~$ tersect samples \
```

```
           tomato.tsi -c \
           "SL2.50ch03:100642:T:G,SL2.50ch06:5001015:A:G"
Sample
S.lyc LA1479
S.pen LA0716
S.hab LYC4
S.hab LA0407
S.hab LA1777
S.hab LA1718
S.hab CGN15792
S.hab PI134418
S.hab CGN15791
S.chm LA2695
```

Set operations

Overview

Tersect can interpret and display the results of set theoretical commands using the `tersect view` command. This is the primary and most flexible functionality of the application and allows the user to construct arbitrarily complex queries. The expected format of a `tersect view` query is as follows:

```
tersect view [options] index.tsi QUERY [REGION1...]
```

Queries

A query is a command interpreted and evaluated by Tersect which (if successful) prints either a list of variants (if the result is a single genome or virtual genome) or a list of genome sample names (if the result is a list of genomes). The simplest query consists of a genome name and prints out the variants belonging to that genome. More advanced queries can contain complex combinations of operations described in the sections below.

Note: The term *virtual genome* refers to a collection of variants not representing a specific genome - for example, the symmetric difference of two genomes (the collection of variants which appear in one but not both of the genomes). Tersect treats these *virtual genomes* the same way it treats "real" genomes so they can be used as operands in nested queries.

Genomes

Genomes can be referred to by their sample name, which is either taken from the header line of the source VCF file or set by the user either manually (see `tersect rename`) or through a tab-delimited name file (see `--name-file` in

`tersect build` and `tersect rename`). A sample name can be of any length and can include any characters (including whitespace) except for single quotes (‘). However, if a sample name includes whitespace, parentheses, or characters used as Tersect operators (-^&|>,\), it has to be surrounded by single quotes (’).

If the query is (or results in) a single genome or virtual genome, the variants contained by that one genome are printed out.

Example:

Print out all the variants belonging to the “S.hab LYC4” genome in the *tomato.tsi* Tersect index file:

```
foo@bar:~$ tersect view \
           tomato.tsi "'S.hab LYC4'"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.hab LYC4'
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  391 .   C   T   .   .   .
SL2.50ch00  416 .   T   A   .   .   .
SL2.50ch00  734 .   T   G   .   .   .
SL2.50ch00  759 .   C   T   .   .   .
SL2.50ch00  771 .   A   G   .   .   .
SL2.50ch00  778 .   T   A   .   .   .
...
```

Note: The sample name had to be surrounded by single quotes because it contains a whitespace character.

Binary operators

Tersect supports four basic binary operators. Each operand has to be a **single** genome. All four operators have the same precedence and are left-associative. You can use parentheses to enforce precedence other than simple left-to-right.

Operator	Name	Usage	Result
&	intersection	GENOME1 & GENOME2	Virtual genome containing variants found in both GENOME1 and GENOME2
|	union	GENOME1 | GENOME2	Virtual genome containing variants found in GENOME1, GENOME2, or both
\	difference	GENOME1 GENOME2	Virtual genome containing variants found in GENOME1 but not in GENOME2

Operator	Name	Usage	Result
^	symmetric difference	GENOME1 ^ GENOME2	Virtual genome containing variants found in GENOME1 or GENOME2 but not in both

The result of a binary operation is treated as a single genome (though it does not have a sample name) called a *virtual genome*, which can be used in further operations.

Examples:

Print out the variants shared by 'S.hua LA1983' and 'S.pim LYC2798':

```
foo@bar:~$ tersect view \
           tomato.tsi "'S.hua LA1983' & 'S.pim LYC2798'"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.hua LA1983' & 'S.pim LYC2798'
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  3235    .   A   G   .   .   .
SL2.50ch00  3277    .   A   G   .   .   .
SL2.50ch00  3873    .   C   T   .   .   .
SL2.50ch00  4083    .   A   G   .   .   .
SL2.50ch00  4112    .   T   G   .   .   .
SL2.50ch00  4314    .   A   C   .   .   .
...
```

Print out the variants which appear in only one of 'S.gal LA1044' or 'S.gal W TS-208':

```
foo@bar:~$ tersect view \
           tomato.tsi "'S.gal LA1044' ^ 'S.gal W TS-208'"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.gal LA1044' ^ 'S.gal W TS-208'
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  362 .   G   T   .   .   .
SL2.50ch00  867 .   G   T   .   .   .
SL2.50ch00  1198    .   G   A   .   .   .
SL2.50ch00  3235    .   A   G   .   .   .
SL2.50ch00  3567    .   T   G   .   .   .
SL2.50ch00  3873    .   C   T   .   .   .
...
```

Print out the variants which appear in 'S.chi CGN15532' but not 'S.chi CGN15530' or 'S.chi W TS-408':

```
foo@bar:~$ tersect view \
```

```
          tomato.tsi \
          "'S.chi CGN15532' \ 'S.chi CGN15530' \ 'S.chi W TS-408'"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.chi CGN15532' \ 'S.chi CGN15530' \ 'S.chi W TS-408'
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  1163    .   C   G   .   .   .
SL2.50ch00  1811    .   C   G   .   .   .
SL2.50ch00  1818    .   C   A   .   .   .
SL2.50ch00  1818    .   C   G   .   .   .
SL2.50ch00  2432    .   G   A   .   .   .
SL2.50ch00  2544    .   A   T   .   .   .
...
```

Note: A more convenient way to conduct the same operation on many genomes is by using functional operators.

Genome list

Instead of individual genomes, Tersect can also operate on lists of genomes. These can be selected using wildcard patterns matching genome sample names, with the most general pattern of a lone wildcard operator (`*`) matching *all* the genomes in the Tersect index file. Individual genomes can also be appended to lists using commas (`,`) or removed from lists using minus signs (`-`).

Genome lists can also be filtered (using the `>` operator) by whether they contain a specified list of variants. The variant format should look as follows: `chromosome:position:ref:alt` where `ref` and `alt` are reference and alternate alleles. Multiple variants can be included, separated by commas (e.g. `chr1:1245:A:G,chr8:5300:T:A`).

Operator	Name	Usage	Result
*	wildcard	PATTERN	Genome list containing all genomes whose sample names match the provided wildcard pattern
,	append	GENOMELIST, GENOME	Genome list containing all genomes in GENOMELIST and GENOME
-	remove	GENOMELIST - GENOME	Genome list containing all genomes in GENOMELIST except GENOME

Operator	Name	Usage	Result
>	superset	GENOMELIST > VARIANTLIST	Genome list containing all genomes in GENOMELIST which contain all variants in VARIANTLIST

Note: Tersect does not distinguish between a genome list which contains only one genome and a single genome. The former can be used in binary operations and the latter can be used in functional operations or in constructing genome lists.

If the query is (or results in) a genome list, the list of their genome sample names is printed out.

Examples:

Print out all the names of genomes which begin with "S.pim":

```
foo@bar:~$ tersect view \
           tomato.tsi "S.pim*"
S.pim P TS-92
S.pim P TS-79
S.pim P TS-77
S.pim P TS-50
S.pim P TS-441
S.pim P TS-440
S.pim P TS-439
S.pim P TS-438
...
```

Print out all the names of genomes which contain an A/G single nucleotide polymorphism at position 828587 in chromosome 7:

```
foo@bar:~$ tersect view \
           tomato.tsi "* > SL2.50ch07:828587:A:G"
S.lyc C TS-97
S.lyc C TS-94
S.pim P TS-79
S.pim P TS-77
S.lyc B TS-68
S.lyc C TS-53
S.pim P TS-50
S.pim P TS-441
...
```

Print out all the names of genomes which contain a G/A SNP at position 1590608 in chromosome 5 and a T/C SNP at position 5230 in chromosome 12, except for 'S.gal LA1401' and those whose names begin with "S.pim":

```
foo@bar:~$ tersect view \
           tomato.tsi \
           "* > SL2.50ch05:1590608:G:A,SL2.50ch12:5230:T:C - ('S.pim*','S.gal LA1401')"
S.lyc C TS-431
S.lyc C TS-430
S.lyc LA1314
```

Functional operators

Functional operators are used to conduct operations on genome lists instead of individual genomes.

Operator	Name	Usage	Result
union() u()	arbitrary union	union(GENOMELIST) u(GENOMELIST)	Virtual genome containing all variants contained in any of the genomes in GENOMELIST
intersect() i()	arbitrary intersection	intersect(GENOMELIST) i(GENOMELIST)	Virtual genome containing all variants which appear in every genome in GENOMELIST

The result of a functional operation is treated as a single genome (though it does not have a sample name).

Examples:

Union of all genomes, containing every variant recorded in the *tomato.tsi* Tersect index file:

```
foo@bar:~$ tersect view tomato.tsi "u(*)"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand=u(*)
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  280 .   A   C   .   .   .
SL2.50ch00  284 .   A   G   .   .   .
SL2.50ch00  316 .   C   T   .   .   .
```

```
SL2.50ch00  323 .   C   T   .   .   .
SL2.50ch00  332 .   A   T   .   .   .
SL2.50ch00  362 .   G   T   .   .   .
...
```

Intersection of all genomes which contain a T/A single nucleotide polymorphism at position 12547 in chromosome 12, containing all variants that are shared by each of those genomes:

```
foo@bar:~$ tersect view \
           tomato.tsi \
           "i(* > SL2.50ch12:12547:T:A)"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand=i(* > SL2.50ch12:12547:T:A)
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  16576   .   T   C   .   .   .
SL2.50ch00  26171   .   G   T   .   .   .
SL2.50ch00  29880   .   A   G   .   .   .
SL2.50ch00  37486   .   T   G   .   .   .
SL2.50ch00  40476   .   G   T   .   .   .
SL2.50ch00  436850  .   A   G   .   .   .
...
```

Print all the variants which appear only in genome S.hab CGN15792. This is achieved by finding the difference of that genome and the union of all genomes except S.hab CGN15792:

```
foo@bar:~$ tersect view \
           tomato.tsi \
           "'S.hab CGN15792' \ u(* - 'S.hab CGN15792')"
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.hab CGN15792' \ u(* - 'S.hab CGN15792')
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
SL2.50ch00  1163    .   C   T   .   .   .
SL2.50ch00  1596    .   G   A   .   .   .
SL2.50ch00  2048    .   G   A   .   .   .
SL2.50ch00  2933    .   G   A   .   .   .
SL2.50ch00  2987    .   A   T   .   .   .
SL2.50ch00  4349    .   C   T   .   .   .
...
```

Regions

By default, queries are executed and results are returned for the entire genome. However, it is possible to selectively execute a query only on a specified region.

The familiar tabix/samtools format `chromosome:beginPos-endPos` is used to specify those regions. The coordinates are one-based and inclusive.

Limiting queries to regions allows for much faster execution since far fewer positions need to be processed and printed, capturing only intervals of interest. This feature makes it possible to use Tersect's flexible queries as a high-performance part of a larger pipeline or the back-end of a highly responsive, interactive application.

Example:

Print a union, that is all the variants appearing either in genome 'S.lyc SG16', 'S.lyc LA1421', or both, from the first 90 kbp of chromosome 2 in the *tomato.tsi* index file:

```
foo@bar:~$ tersect view \
           tomato.tsi \
           "'S.lyc SG16' | 'S.lyc LA1421'" SL2.50ch02:1-90000
##fileformat=VCFv4.3
##tersectVersion=0.11.0
##tersectCommand='S.lyc SG16' | 'S.lyc LA1421'
##tersectRegion=SL2.50ch02:1-90000
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
SL2.50ch02      204     .       A       G       .       .       .
SL2.50ch02      255     .       TCC     TCCC    .       .       .
SL2.50ch02      255     .       TCC     TCCCC   .       .       .
SL2.50ch02      2382    .       G       A       .       .       .
SL2.50ch02      13383   .       G       A       .       .       .
SL2.50ch02      21752   .       C       T       .       .       .
SL2.50ch02      24538   .       T       C       .       .       .
SL2.50ch02      29276   .       G       T       .       .       .
SL2.50ch02      71245   .       A       C       .       .       .
SL2.50ch02      73326   .       C       T       .       .       .
SL2.50ch02      86236   .       C       A       .       .       .
SL2.50ch02      86601   .       A       G       .       .       .
SL2.50ch02      86635   .       T       A       .       .       .
SL2.50ch02      86695   .       T       C       .       .       .
SL2.50ch02      86769   .       G       A       .       .       .
SL2.50ch02      87079   .       T       A       .       .       .
```

Supplementary material S2
Tersect Indexing Documentation

Version 0.12.0 (16 April 2019)

1 TERSECT INDEX FORMAT

Tersect relies on constructing index / database files to enable it to execute its high-performance queries. The files are in a custom binary format and use the "tsi" filename extension (standing for **t**er**s**ect **i**ndex) by default.

The first fourteen bytes in the index files encode an ASCII representation of the TSI file format version used. This is a C-string (i.e. 13 characters followed by a null terminator) and "TersectDB 0.2" is currently the only valid value. This is to allow for the correct interpretation of the header that follows, and the rest of the data contained in the index file.

The TSI header contains the following information:

- Database size in bytes (64-bit unsigned integer)
- Word size used by the database (16-bit unsigned integer)
- Offset of the chromosome list data structure (64-bit unsigned integer)
- Number of chromosomes (32-bit unsigned integer)
- Offset of the genome list data structure (64-bit unsigned integer)
- Number of genomes (32-bit unsigned integer)
- Offset of the free list data structure (64-bit unsigned integer)

While much of the data they contain is compressed, Tersect index files can still be quite large (several gigabytes and more on real data sets). Rather than fully parsing them, Tersect uses them as memory-mapped files. The data structures they contain refer to each other through offsets from the start of the index file. Tersect translates these offsets to pointers based on the mapping location, casting the data structures stored in the file into a representation used elsewhere in the application. The former 'internal' data structures are described in the *tersect_db_internal.h* header, while their 'public' interfaces used elsewhere can be seen in the *tersect_db.h* header.

Tersect manages memory within the index file using a simple free list and first-fit allocation, expanding the file in page-sized chunks if not enough space is available. Note that the index file is generally intended to be created in a single batch process (in which case there is little fragmentation and no wasted space) with very little later modification, such as renaming samples.

2 INDEX FILE CONSTRUCTION

To construct an index file, Tersect uses a custom parser to merge the contents of input VCF files into per-chromosome lists of alleles. This is done using a priority queue algorithm that includes a normalization and local sorting step on each of the input files to ensure the variant alleles are stored in a normalized, unambiguous order (sorted first by position, then alphabetically by the allele base sequences). Single nucleotide polymorphisms are encoded using numeric codes for each reference/alternate base combination, while larger variants have their sequences stored as strings allocated in the index file, with the variant list recording the string location offset.

Note that the elements of the lists are individual alleles rather than sites or calls as in the VCF format. Variants at multiallelic sites are split into separate entries.

Parallel to the per-chromosome variant lists, Tersect also builds per-sample bit arrays for each chromosome. These encode the presence or absence of each variant in specific samples and are directly parallel to the per-chromosome variant lists (so the presence of variant number N in the variant list is encoded by bit number N in the bit array). The bit arrays are built and read using a compressed representation, without an explicit compression/decompression step. The algorithm is described in more detail in the following section.

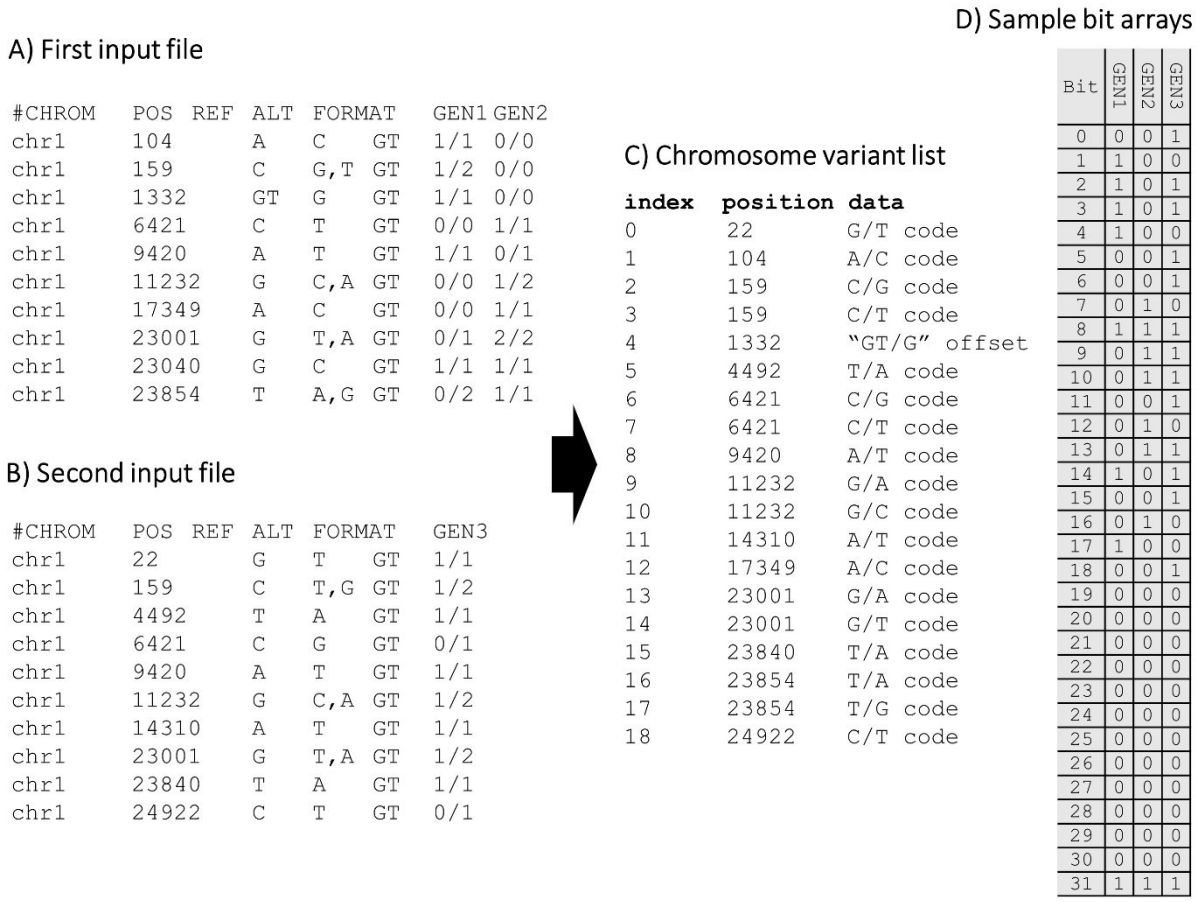


Figure 1: ***Tersect index file construction diagram.*** *Parts A) and B) show the contents of example VCF input files (metadata and certain columns were omitted). The first input file contains data for two samples (GEN1 and GEN2) and the second file contains data for a single sample (GEN3). All alleles contained in a chromosome are stored in a single list as seen in part C). Membership of individual alleles in each of the samples is encoded in bit arrays as seen in part D), which shows a 32-bit word for the sake of simplicity (Tersect uses 64-bit words by default). The most significant bit is set for all three bit arrays, indicating that the specific word shown is a* ***literal*** *word (as opposed to a* ***fill*** *word – these terms are explained in section 3 below). Note that the indices in the chromosome variant table and the sample bit arrays match – the lists are parallel.*

3 COMPRESSION

Per-sample presence or absence of specific variants of a chromosome is encoded in bit arrays using a variant of the Word-Aligned Hybrid (WAH) compression algorithm. The primary data structure is stored as a simple array of 64-bit words corresponding to the entire length of the chromosome. Each of the words is either a **literal** word or a **fill** word; this distinction is indicated by the most significant bit of each word, which is set for literal words and unset for fill words. This leaves 63 bits for other data.

Literal words use their 63 bits to store the presence (set bit) or absence (unset bit) of up to 63 successive variants. Fill words store the length of a run of absent variants in multiples of 63, a type of run-length

encoding (RLE). Thus, a fill word containing the (decimal) number 1 indicates a run of 63 absent variants, number 2 indicates a run of 126 absent variants, and so on.

Note that recording runs of empty *words* rather than empty *bits*, while wasteful of space, keeps the bits in literal words aligned so that no bit shifting is required to conduct binary operations on variants at the same positions. This trades space for execution speed.

In classical WAH compression, fill words can be used to indicate runs of either set or unset bits (and potentially other patterns), with the type of fill word being indicated by successive most significant bits. In the Tersect implementation this was simplified and limited to only runs of unset bits for several reasons. The arrays used for indicating variant contents are very sparse and runs of more than 63 set bits are rare, making the improvement in compression had they been included minor. At the same time, limiting fill word metadata to a single bit flag set to 0 means no further flag checks or manipulations are necessary and the value stored in the word can be used directly as an integer representing the run length. This simplifies the code and yields an improvement in execution speed.

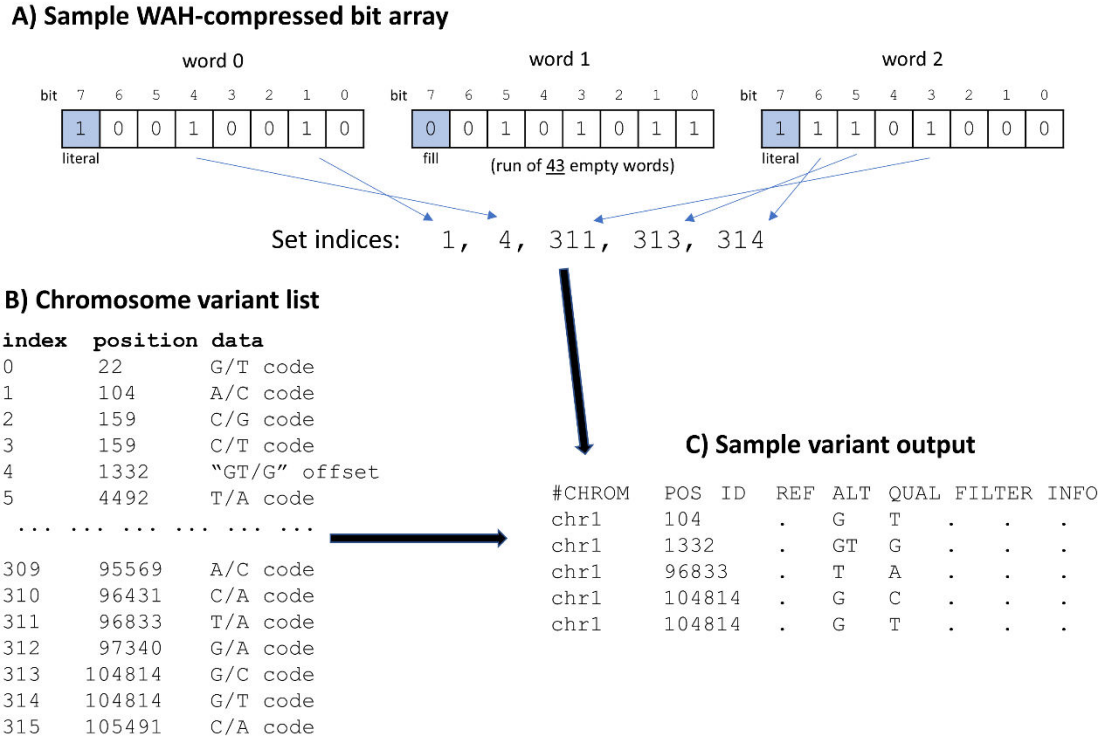


Figure 2: Diagrammatic example of WAH compression and variant retrieval by Tersect. *Three words (part A) encode allele contents for 315 successive alleles stored in the chromosome variant list shown in part B. The literal words encode the indices of variants present in a sample, while the fill word records the length of a run of empty words (each corresponding to seven absent alleles). The stored binary value is 0b0101011 (decimal 43). With seven alleles per word, this can be used to advance the index indicator of the variant list by 7 x 43 = 301 positions when the bit array is traversed. Note that, while for the sake of simplicity the example uses 8-bit words, Tersect uses 64-bit words by default.*

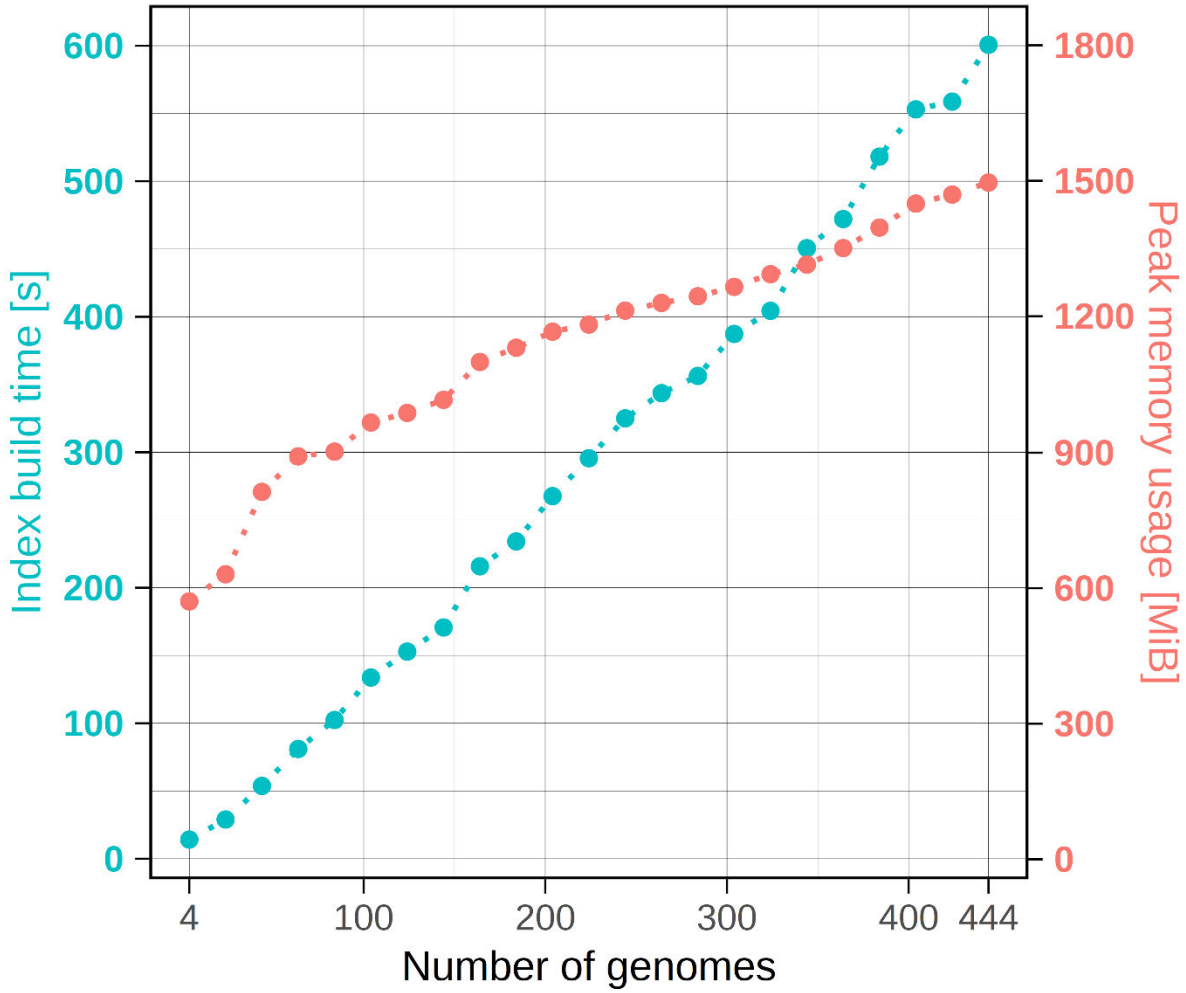
Tersect uses 64-bit words by default, but this is a value which can be changed at compile-time. Using a word size matching the processor word size (64-bit in most architectures common today) is generally the best choice from a performance standpoint, as it makes it possible to make better use of SIMD extended instruction sets to speed up operations on bit arrays. However, smaller word sizes may yield superior compression due to higher data granularity: with 64-bit words one can only save a word of memory when at least 126 successive word-aligned variants are absent (and another word for each further 63 such variants), while with 32-bit words a word is saved starting with the 62nd absent variant

(and another is saved for each further 31 such variants). Still, the proportion of metadata (the literal/fill word flag) also rises as the word size grows smaller: for 8-bit words, where metadata takes up 12.5% of the storage, the memory use actually increases!

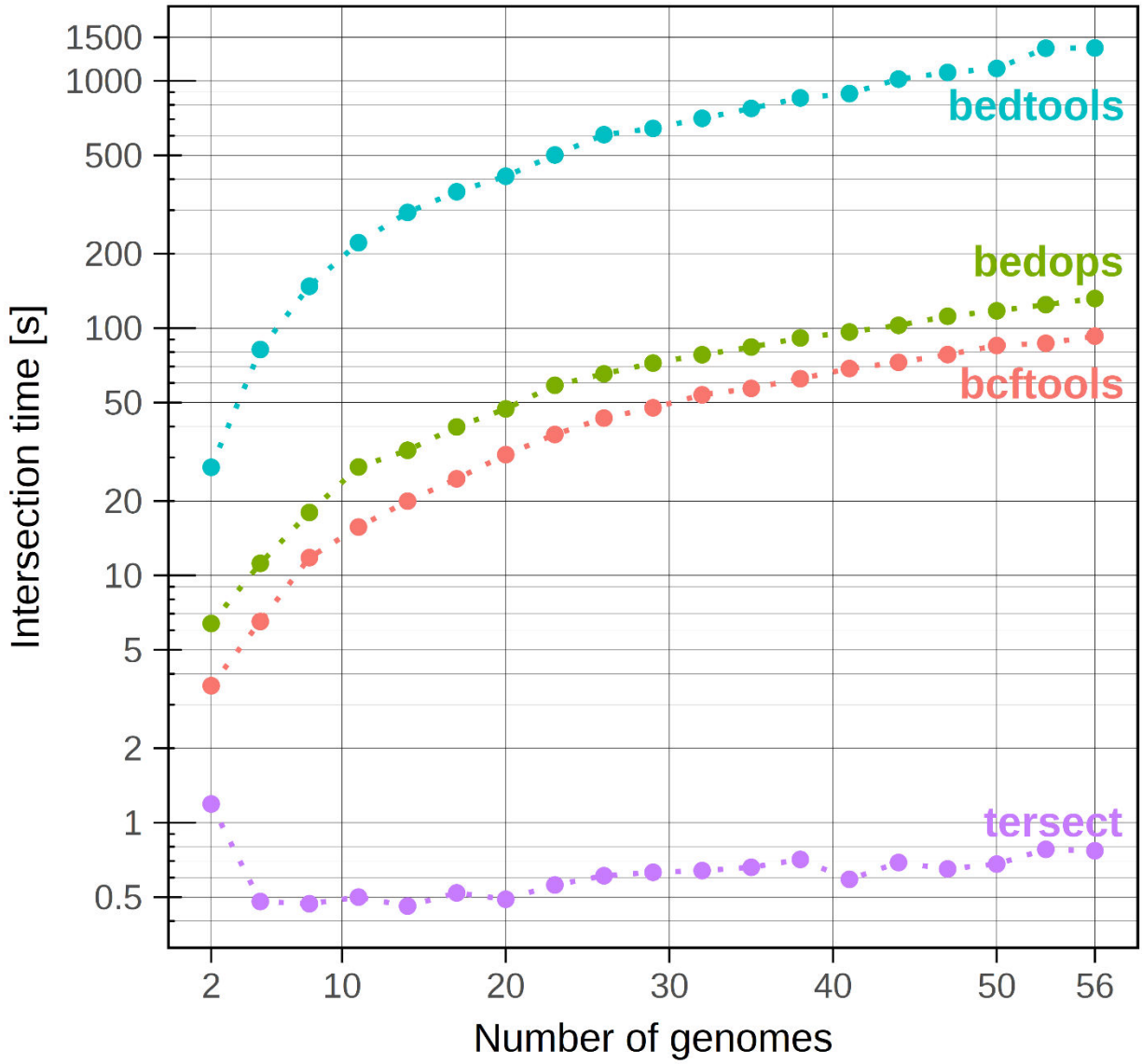
Another consequence of changing the word size is that index files generated by Tersect compiled with a certain word size are not compatible with Tersect compiled with a different word size. This is why the default word size is set to 64-bits instead of varying based on architecture. Advanced users are free to fine-tune this at compile-time, but they will not be able to use the example data sets provided with tutorials.

Supplementary material S3

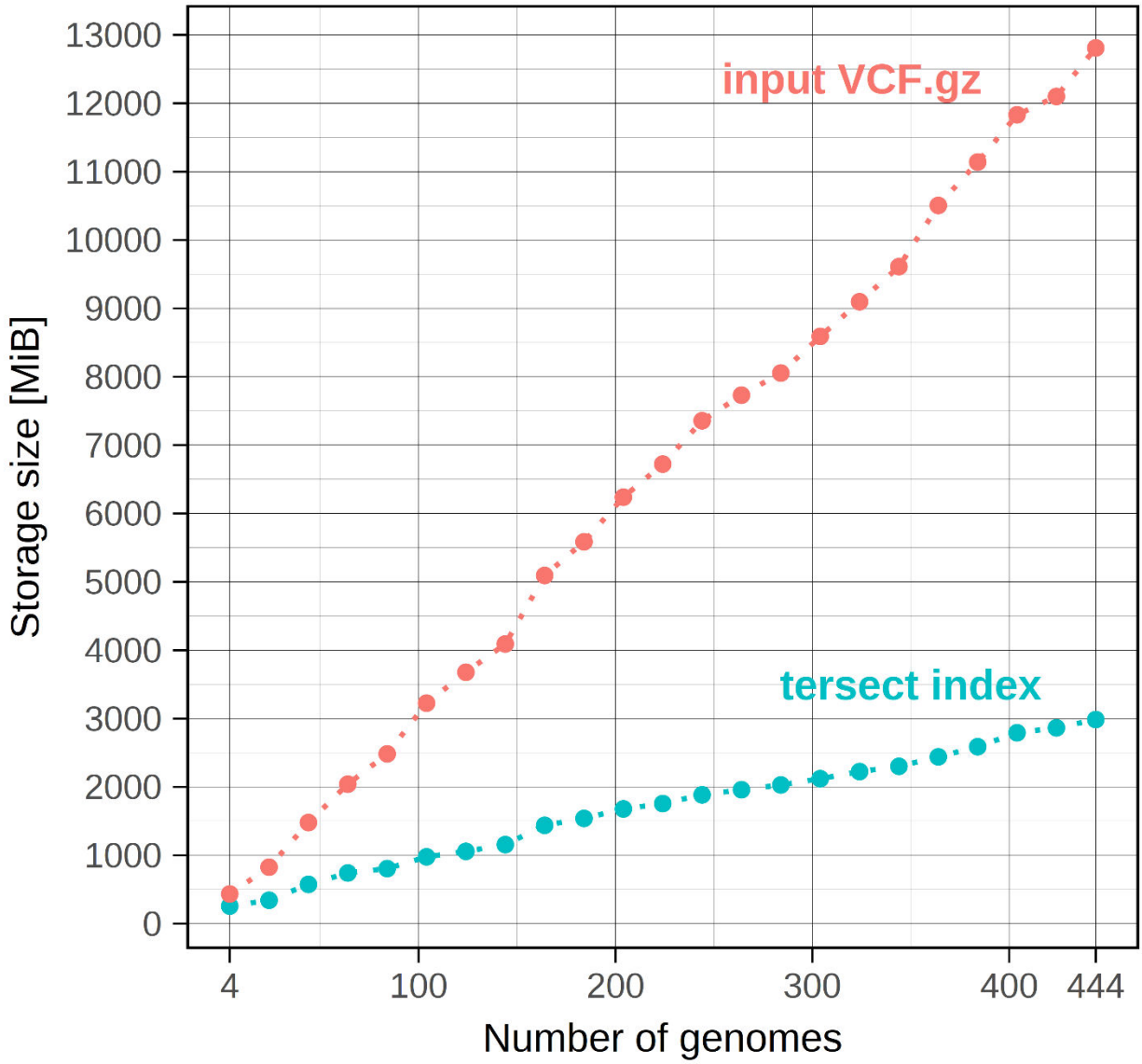
Supplementary Figures and Tables



Supplementary Figure 1. Tersect index build time and peak memory usage. It should be noted that inclusion time per genome varied significantly due to different variant content per genome, evident in the shape of the line. This is also evident in Supplementary Figure 3, which traces a very similar path for the input data. The source genome list was shuffled to minimize this variation. The peak memory usage is defined as the maximum resident set size. See Supplementary Table 3 for numeric results.



Supplementary Figure 2. Benchmarking results for the intersections of subsets of 56 *Solanum pimpinellifolium* genomes. This is a wild species of tomato closely related to the *S. lycopersicum*, the cultivated tomato and the most numerous wild tomato species in the source data sets. The shared variants identified through intersection represent alleles typical of *S. pimpinellifolium* as compared to the cultivated tomato reference genome. Note that for Tersect, the execution time is the highest for the smallest input size (two genomes). This is because, as is typical for intersection, the output variant set becomes smaller the more genomes are included. For only two genomes printing the result takes much longer than computing it. See Supplementary Table 2 for numeric results.



Supplementary Figure 3. Size of input data and generated Tersect index files. Note that the sizes of individual per genome data sets vary with the number of variants they contain. See Supplementary Table 3 for numeric results.

Supplementary Table 1. Private variant identification benchmark results.

	Private variant identification time [seconds]			
Number of genomes	**Tersect Version 0.12.0**	**BCFtools Version 1.9**	**BEDTools Version 2.27.1**	**BEDOPS Version 2.4.35**
4	3.03	12.26	108.37	9.42
24	2.98	28.45	362.75	33.49
44	2.53	56.69	756.15	99.56
64	2.84	84.13	1167.93	139.8
84	2.94	106.18	1555.52	191.23
104	2.59	142.92	2152.15	273.26
124	2.72	173.96	2339.3	325
144	3.13	212.22	2786.44	384.52
164	2.92	276.66	3506.33	510.79
184	3.1	318.23	3694.72	581.05
204	3.2	361.42	4246.82	669.34
224	3.42	397.74	4652.93	715
244	3.59	447.7	5199.98	788.68
264	3.96	474.7	5654.76	896.68
284	4.13	515.08	5804.88	971.73
304	4.42	553.76	6273.72	1024.31
324	4.46	609.85	6460.3	1105.54
344	4.88	654.08	7038.13	1169.67
364	4.95	712.29	7871.98	1273.92
384	5.3	766.78	8232.68	1409.26
404	5.33	844.31	8833.28	1562.77
424	5.66	900.02	9001.77	1665.46
444	5.72	956.39	9463.02	1753.81

Supplementary Table 2. Intersection benchmark results.

	Intersection time [seconds]			
Number of genomes	**Tersect Version 0.12.0**	**BCFtools Version 1.9**	**BEDTools Version 2.27.1**	**BEDOPS Version 2.4.35**
2	1.19	3.58	27.37	6.39
5	0.48	6.51	81.94	11.19
8	0.47	11.8	147.65	17.97
11	0.5	15.69	221.64	27.45
14	0.46	19.97	293.82	32.06
17	0.52	24.59	356.1	39.86
20	0.49	30.79	411.51	47.15
23	0.56	37.14	501.51	58.73
26	0.61	43.26	605.98	65.34
29	0.63	47.62	642.08	72.23
32	0.64	53.74	705.53	78.06
35	0.66	57.06	773.71	83.86
38	0.71	62.44	853.04	91.27
41	0.59	68.66	888.3	96.49
44	0.69	72.71	1016.69	102.64
47	0.65	78.17	1081.67	111.71
50	0.68	85.04	1122.27	117.5
53	0.78	86.84	1355.56	124.44
56	0.77	92.85	1358.65	131.78

Supplementary Table 3. Tersect index build metrics.

Number of genomes	Input VCF.gz size [MiB]	Output Tersect index size [MiB]	Tersect index build time [s]	Tersect index build peak memory usage [MiB]
4	434	257	14.18	570
24	826	342	28.98	630
44	1479	574	53.76	812
64	2040	741	81.05	891
84	2483	805	102.43	901
104	3225	977	133.75	966
124	3677	1057	152.87	987
144	4091	1156	170.7	1016
164	5093	1438	215.84	1100
184	5585	1539	234.18	1131
204	6238	1678	267.63	1167
224	6722	1756	295.59	1183
244	7356	1884	325	1214
264	7730	1959	343.6	1231
284	8054	2029	356.33	1245
304	8590	2120	387.3	1266
324	9097	2224	404.35	1294
344	9610	2301	450.6	1315
364	10505	2440	472.07	1352
384	11140	2588	518.17	1397
404	11831	2792	553.02	1450
424	12099	2864	558.73	1470
444	12809	2985	600.68	1497